Feature Encoding or Variable Encoding

In many practical data science projects, the data set will contain categorical variables. These variables are typically stored as text values. Since machine learning is based on mathematical equations, it would cause a problem when we keep categorical variables as is. Many algorithms support categorical values and algorithms that do not support categorical values, in that case, are left with encoding methodologies.

Categorical Feature Types

1.Binary:

- Examples: Yes, No, True, False

2.Ordinal: Specific ordered Groups.

- Examples: low, medium, high, cold, hot, lava Hot

3.Nominal: Unordered Groups.

- Examples: cat, dog, tiger, pizza, burger, coke

Encoding Techniques

- Label Encoding
- Ordinal Encoding
- One-Hot Encoding
- Binary Encoding
- Frequency Encoding
- Mean Encoding

1. Label Encoding

In this encoding each category is assigned a value from 1 to N where N is the number of category in the column. This encoding can be used for nominal data type. The major issue using this encoding is their is no relation or order in the category but algorithm thinks their should be an order or relation between these category.

Example: Cold,Hot,Very Hot,Warm… these categories are converted to 0,1,2,3... and ML model thinks that their is relation 0<1<2<3....

In [ ]:
```
# Importing packages
import pandas as pd

# Creating a dictionary for some records.
data_dict = {'name':['ABC','XYZ','PQR'],
             'degree':['Btech','MBA','btech'],
             'gender':['Male','Female','Female'],
             'Performance':['Excellent','Very Good','Bad'],
             'percentage':[70.5,80,80.3],
             'Experience':[1,2,0],
             'Promotion':['yes','yes','No']}


data = pd.DataFrame(data_dict)
```

In [ ]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
label_data = data.copy()
label_data['degree'] = le.fit_transform(label_data['degree'])
label_data.head()
```

Out[ ]:

	name	degree	gender	Performance	percentage	Experience	Promotion
0	ABC	0	Male	Excellent	70.5	1	yes
1	XYZ	1	Female	Very Good	80.0	2	yes
2	PQR	2	Female	Bad	80.3	0	No

- To overcome the Disadvantage of Label Encoding as it considers some hierarchy in the columns which can be misleading to nominal features present in the data. we can use **One-Hot Encoding strategy.**

2. One-Hot Encoding

This is the best approach for dealing nominal data as well as ordinal data. In this encoding new column will be created for every category in the column.For each column gets value `1` if that row contains column value else `0` .

In this approach the no of columns created will be equal to number of category in the column. This will make memory problems when the data set has high features.

In [ ]:
```
from sklearn.preprocessing import OneHotEncoder

onehot_data = data.copy()
enc = OneHotEncoder()

# tranforming the column after fitting
enc = enc.fit_transform(onehot_data[['degree']]).toarray()

# converting arrays to a dataframe
encoded_colm = pd.DataFrame(enc,columns = set(list(onehot_data
['degree'])))

# concating dataframes
onehot_data = pd.concat([onehot_data, encoded_colm], axis = 1)

# removing the encoded column.
onehot_data = onehot_data.drop(['degree'], axis = 1)
onehot_data.head()
```

Out[ ]:

	name	gender	Performance	percentage	Experience	Promotion	Btech	M
0	ABC	Male	Excellent	70.5	1	yes	1.0	
1	XYZ	Female	Very Good	80.0	2	yes	0.0	
2	PQR	Female	Bad	80.3	0	No	0.0	

From above One-hot Endoing is applied to degree column it then created three columns that is MBA,Btech,btech which values in degree column.

One-hot encoding approach eliminates the order but it causes the number of columns to expand vastly. we can use Frequency Encoding strategy

Frequency Encoding strategy

In this approach encoding is done by considering the frequency distribution. It is calculated by no of times category occured / Total number of categories

In [ ]:
```
# grouping by frequency
frequency_data = data.copy()
fq = frequency_data.groupby('degree').size()/len(frequency_dat
a)

# mapping values to dataframe
frequency_data['degree'] = frequency_data['degree'].map(fq)

frequency_data.head(10)
```

Out[ ]:

	name	degree	gender	Performance	percentage	Experience	Promotion
0	ABC	0.333333	Male	Excellent	70.5	1	yes
1	XYZ	0.333333	Female	Very Good	80.0	2	yes
2	PQR	0.333333	Female	Bad	80.3	0	No

Ordinal Encoding

- This approach can be used for ordinal datatype. This encoding ensures that a variable retains the ordinal nature.
- Example :

  temp={'cold':0,'warm':1,'hot':2,'veryhot':3}

  It defines some meaning to the categories.

In [ ]:
```
# we can use map function
# create mapping values
performance_map = {'Excellent':3,'Very Good':2,'Bad':1}

ordinal_data = data.copy()
# map to the column
ordinal_data['Performance'] = ordinal_data['Performance'].map
(performance_map)

ordinal_data.head()
```

Out[ ]:

	name	degree	gender	Performance	percentage	Experience	Promotion
0	ABC	Btech	Male	3	70.5	1	yes
1	XYZ	MBA	Female	2	80.0	2	yes
2	PQR	btech	Female	1	80.3	0	No

Binary Encoding

- This approach overcomes the disadvantage of one-hot encoding. This approach first convert the category into numbers starting from 1 to N where N is number of categories in the column. And then each number is converted into binary. For example 3 becomes 011 , 4 becomes 100.
- Each digit in the binary will form seperate column. For 100 category one-hot encoding creates 100 columns but in binary it creates 7 columns.

In [ ]:
```
!pip install category_encoders
```

```
Collecting category_encoders
  Downloading category_encoders-2.4.1-py2.py3-none-any.whl (80
kB)
     |████████████████████████████████| 80 kB 5.1 MB/s
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib
/python3.7/dist-packages (from category_encoders) (1.21.6)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/
python3.7/dist-packages (from category_encoders) (1.4.1)
Requirement already satisfied: pandas>=0.21.1 in /usr/local/li
b/python3.7/dist-packages (from category_encoders) (1.3.5)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/loca
l/lib/python3.7/dist-packages (from category_encoders) (0.10.
2)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/
python3.7/dist-packages (from category_encoders) (0.5.2)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/lo
cal/lib/python3.7/dist-packages (from category_encoders) (1.0.
2)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/
local/lib/python3.7/dist-packages (from pandas>=0.21.1->catego
ry_encoders) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/
python3.7/dist-packages (from pandas>=0.21.1->category_encoder
s) (2022.1)
Requirement already satisfied: six in /usr/local/lib/python3.7
/dist-packages (from patsy>=0.5.1->category_encoders) (1.15.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/lo
cal/lib/python3.7/dist-packages (from scikit-learn>=0.20.0->ca
tegory_encoders) (3.1.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/
python3.7/dist-packages (from scikit-learn>=0.20.0->category_e
ncoders) (1.1.0)
Installing collected packages: category-encoders
Successfully installed category-encoders-2.4.1
```

In [ ]:
```
from category_encoders import BinaryEncoder

binary_data = data.copy()

encoder = BinaryEncoder(cols =['name'])

# tranforming the column after fitting
newdata = encoder.fit_transform(binary_data['name'])

# concating dataframe
binary_data = pd.concat([binary_data, newdata], axis = 1)

# dropping old column
binary_data = binary_data.drop(['name'], axis = 1)
binary_data.head(10)
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_test
ing.py:19: FutureWarning: pandas.util.testing is deprecated. U
se the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

Out[ ]:

	degree	gender	Performance	percentage	Experience	Promotion	name_0
0	Btech	Male	Excellent	70.5	1	yes	0
1	MBA	Female	Very Good	80.0	2	yes	1
2	btech	Female	Bad	80.3	0	No	1

Mean Encoding or Target Encoding

- In this apprroach we will calculate the mean of target variable for each category, and replace category value with mean value. From below for category `s1` the values of target variable is ```1,1,1,0```` then mean will be 3/4 = 0.75

In [ ]:
```
# importing libraries
import pandas as pd

# creating dataset
data={'SubjectName':['s1','s2','s3','s1','s4','s3','s2','s1','s2','s4','s1'],
      'Target':[1,0,1,1,1,0,0,1,1,1,0]}

df = pd.DataFrame(data)

print(df)

# Here we apply mean for target variable for each class.
# Example For s1 class has target values is 1,1,1,0 so average is 0.75.
# and s1 is replaced by 0.75
Mean_encoded_subject = df.groupby(['SubjectName'])['Target'].mean().to_dict()

df['SubjectName'] =  df['SubjectName'].map(Mean_encoded_subject)

print(df)
```

```
   SubjectName  Target
0           s1       1
1           s2       0
2           s3       1
3           s1       1
4           s4       1
5           s3       0
6           s2       0
7           s1       1
8           s2       1
9           s4       1
10          s1       0
    SubjectName  Target
0      0.750000       1
1      0.333333       0
2      0.500000       1
3      0.750000       1
4      1.000000       1
5      0.500000       0
6      0.333333       0
7      0.750000       1
8      0.333333       1
9      1.000000       1
10     0.750000       0
```